ADULT INCOME

PREDICTION

UTS 2020 Autumn Data Analytics Assignment 3

Abstract

In this assignment, I try to build classifiers to use on the provided dataset. Classifiers should predict the Annual income of the adults.

Bamshad Behinaein

13708134@student.uts.edu.au

Introduction

This report is a brief description of data pre-processing and classification of a given dataset. This dataset contains 16 column information about 38000 individuals. After building the classifier, I have to use that to predict the salary of the test data frame and submit that to Kaggle. The test dataset contains 10000 rows without the last column, which is salary. I need to pre-process these data frames to make them ready for the classification.

After pre-processing the train and test datasets, I need to build the classifier. Several classifiers have been tested for this assignment which will be explained in this report. After choosing the right classifier, I trained it and gave the pre-processed test dataset to predict the probability of the salary. It should predict if the salary is greater or lower than $50k.

For this assignment, I used the Python programming language. The tools that are used are Pandas, Numpy, Scikit-learn, and Matplotlib. These tools are available at Anaconda Which is an open-source Python distribution for machine learning and data science. My environment was Jupyter Notebook.

Data Mining Problem

The provided dataset is unpre-processed and is not ready to use in classifiers. Here I try to state some of these problems.

The main issue of the dataset is categorical values. The classifier cannot process strings, and it needs numbers to apply algorithms on them. In the pre-processing step, this problem should be solved. In the following screenshot, you can see types of the values of the training dataset. As you can see, there are 6 integer Attribute types (minus ID) and 9 object types. The object is a string, and it is not useful for training the machine learning model.

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38000 entries, 0 to 37999
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ID                    38000 non-null  int64
 1   Age                   38000 non-null  int64
 2   Employment class      38000 non-null  object
 3   Fnlwgt                38000 non-null  int64
 4   Education level       38000 non-null  object
 5   Education years       38000 non-null  int64
 6   Marital status        38000 non-null  object
 7   Occupation            38000 non-null  object
 8   Relationship status   38000 non-null  object
 9   Race                  38000 non-null  object
 10  Sex                   38000 non-null  object
 11  Capital gain          38000 non-null  int64
 12  Capital loss          38000 non-null  int64
 13  Work hours per week   38000 non-null  int64
 14  Native country        38000 non-null  object
 15  Salary                38000 non-null  object
dtypes: int64(7), object(9)
memory usage: 4.6+ MB
```

Figure 1 - dataset info

Another issue is data is spread for some attributes. For example, the minimum value of the “Fnlwgt” attribute is 1.228500e+04, and the maximum is 1.455435e+06. There is same problem for “Age”, “Capital gain “,” Capital loss”, and “Work hours per week”. To solve this problem, I need to use data normalization, and you can see that in data pre-processing and transformation step. The following figure is a simple description of the statistics of the dataset.

```
In [7]: dataset.describe()
```

Out[7]:

	ID	Age	Fnlwgt	Education years	Capital gain	Capital loss	Work hours per week
count	38000.000000	38000.000000	3.800000e+04	38000.000000	38000.000000	38000.000000	38000.000000
mean	23987.651500	38.525711	1.893863e+05	10.069842	1098.063289	87.803211	40.483026
std	13840.226031	13.645989	1.052969e+05	2.553744	7478.191228	403.923362	12.346755
min	1.000000	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	12012.750000	28.000000	1.177460e+05	9.000000	0.000000	0.000000	40.000000
50%	24003.500000	37.000000	1.783700e+05	10.000000	0.000000	0.000000	40.000000
75%	35947.250000	47.000000	2.363280e+05	12.000000	0.000000	0.000000	45.000000
max	48000.000000	90.000000	1.455435e+06	16.000000	99999.000000	4356.000000	99.000000

Figure 2 - dataset description

Data Pre-processing and transformations

As mentioned in the above part, we have 2 datasets that we need to prepare them which are train and test data frames. I decided to open 2 different notebooks for each one of them to be able to do particular pre-processing on them.

First, I explain the preparation for the training dataset. The necessary tools for this part are Numpy and Pandas. As you can see, I imported them from Python's library. Next step is to load the dataset. The frame contains categorical, ordinal, interval, and ratio attributes that we need to process them.

```
import pandas as pd
import numpy as np
```

```
dataset = pd.read_csv("train.csv")
dataset
```

	ID	Age	Employment class	Fnlwgt	Education level	Education years	Marital status	Occupation	Relationship status	Race	Sex	Capital gain	Capital loss	Work hours per week	Native country	Salary
0	40947	50	Private	104729	HS-grad	9	Divorced	Machine-op-inspct	Other-relative	White	Female	0	0	48	United-States	<=50K
1	17139	27	Private	138705	HS-grad	9	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	53	United-States	<=50K
2	29557	30	Private	144593	HS-grad	9	Never-married	Other-service	Not-in-family	Black	Male	0	0	40	?	<=50K
3	10344	40	Private	181015	HS-grad	9	Separated	Other-service	Unmarried	White	Female	0	0	47	United-States	<=50K
4	33206	52	Private	110563	Some-college	10	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	40	United-States	>50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 3 - import tools and read dataset

The pre-processing of the data starts from binarizing selected attributes. This action lets the classifier to process the categorical and ordinal values. I used the Dummies class of the Pandas to binarize the columns.

```
dftest = pd.get_dummies(dataset, columns=["Salary","Sex","Race","Marital status","Employment class","Occupation"])
```

Figure 4 - binarizing (dummies)

To simplify the marital status, I decided to use married and unmarried states to define the attribute. To do that, I sum up different forms of married dummies and delete unmarried dummies. 1 represents married status and 0 for single individuals.

```
dftest['Married'] = dftest.loc[:,['Marital status_ Married-AF-spouse','Marital status_ Married-civ-spouse',
                                  'Marital status_ Married-spouse-absent']].sum(axis=1)
```

Figure 5 - marital status

I used min-max normalization for “Age”, “Education years”, “Fnlwgt”, and “Work hours per week” attributes. To make that happen, I used simple math and added a new column for each normalized attribute.

```
dftest['Age_normalize'] = (dftest['Age'] - dftest['Age'].min()
                           )/(dftest['Age'].max() - dftest['Age'].min())
dftest['Education_years_n'] = (dftest['Education years'] - dftest['Education years'].min()
                           )/(dftest['Education years'].max() - dftest['Education years'].min())
dftest['Fnlwgt_normalize'] = (dftest['Fnlwgt'] - dftest['Fnlwgt'].min()
                           )/(dftest['Fnlwgt'].max() - dftest['Fnlwgt'].min())
dftest['Work_hours_per_week_n'] = (dftest['Work hours per week'] - dftest['Work hours per week'].min()
                           )/(dftest['Work hours per week'].max() - dftest['Work hours per week'].min())
```

Figure 6 - normalizing

For “Capital gain” and “Capital loss” attributes, I decided to mixed them by a simple subtraction and named the new column as “Capital”. After mixing them, I used Z-score normalization to standardize “Capital” value.

```
Capital = dftest['Capital gain'] - dftest['Capital loss']
dftest['Capital_n'] = (Capital - Capital.mean())/(Capital.std())
```

Figure 7 - normalizing Capital

Since the majority of the “Native country” values were “United-States”, I decided to use one attribute as “US_born”, and replace “United-States” values with 1, and other ones with 0. It let us have a binarize attribute from “Native country” column.

```
dftest['US_born'] = np.where(dftest["Native country"].str.contains("United-States"), 1, 0)
dftest = dftest.drop(['Native country'], axis=1)
```

Figure 8 - binarizing Native country

Now it is time to drop redundant attributes. Since we got dummies for many columns, we have some extra attributes that should drop. I have to mention that for “Race” attribute because the majority of this column was “White”, I decided to use only one binarize column that tells us is the person “White” or not.

```
dftest = dftest.drop(['Employment class_ Never-worked','Marital status_ Divorced','Marital status_ Married-AF-spouse
                      'Marital status_ Married-civ-spouse','Marital status_ Married-spouse-absent',
                      'Marital status_ Never-married','Marital status_ Separated',
                      'Marital status_ Widowed','Salary_ >50K','Sex_ Male',
                      'Race_ Amer-Indian-Eskimo','Race_ Asian-Pac-Islander','Race_ Black',
                      'Race_ Other','ID','Education level','Relationship status','Age','Fnlwgt',
                      'Education years','Work hours per week','Capital gain','Capital loss'], axis=1)
```

Figure 9 - drop useless columns

This step is a simple procedure to move greater than $50K salaries to the end. I have to mention that this attribute means greater than $50K, not the other way.

```
dftest['Salary=>50K'] = dftest['Salary_ <=50K']
dftest = dftest.drop(['Salary_ <=50K'], axis=1)
```

Figure 10 - fixing place of Salary

Now the dataset is ready and I extracted it for the classificatilion.

```
dftest.to_csv('readydata1.csv', index=False)
```

Figure 11 - saving the file

For the test data set, I did all of the above procedure except one part that I explain below. First of all I read the test data frame.

```
dataset = pd.read_csv("test-pub.csv")
dataset.head()
```

	ID	Age	Employment class	Fnlwgt	Education level	Education years	Marital status	Occupation	Relationship status	Race	Sex	Capital gain	Capital loss	Work hours per week	Native country
0	21990	24	Private	138768	Assoc-acdm	12	Never-married	Handlers-cleaners	Own-child	White	Male	0	0	30	United-States
1	6594	28	Local-gov	214881	HS-grad	9	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States
2	18525	37	Private	177181	HS-grad	9	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States
3	23747	66	Private	350498	Some-college	10	Married-civ-spouse	Sales	Husband	White	Male	0	0	28	United-States
4	37752	34	Self-emp-not-inc	56460	HS-grad	9	Married-civ-spouse	Farming-fishing	Wife	White	Female	0	2179	12	United-States

Figure 12 - preparation of test dataset

Because we should predict the salary, we do not have that column for the test dataset. Also, we should keep the “ID” attribute for the record. The following figure is the binarizing of the test dataset.

```
dftest = pd.get_dummies(dataset, columns=["Sex","Race","Marital status","Employment class","Occupation"])
```

Figure 13 - binarizing test dataset

All of the other procedure is same as the training dataset, so I did not explain them again. Now is time to extract the test data frame.

```
dftest.to_csv('readydatatest1.csv', index=False)
```

Figure 14 - Saving test dataset

At this point, pre-processing is finished, and the data is ready for the classification. To reach this point, I prepared the data many times and tested it with the classifiers to attain the best way to pre-process the data.

Solving the Problem

This part is about how I find the best classifier to predict adults’ income. We have several classifiers and machine learning algorithms to predict models, and deciding the most accurate one for the model is a challenge. In figure 15 you can see tools that I imported for the classification. The classifiers that I tested them are Logistic Regression, Decision Tree Classifier, Random Forest Classifier, K-neighbours Classifier, Linear Discriminant Analysis, Gaussian NB, and Support Vector Classifier. The total number of classifiers are 7.

```
import pandas as pd
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn import preprocessing
```

Figure 15 - import tools

First of all, I imported the pre-processed training dataset to build the classifier. I prepared the data in previous step.

```
dataset = pd.read_csv("readydata1.csv")
```

Figure 16 - read the train dataset

Classifiers need the data to be as arrays. This step is supposed to prepare the form of data frame for classifiers. For “X” variable need all the values except the last column, which is the target, and “Y” variable is the target. We also need to scale the training data. The pre-processing class was imported from the SKlearn.

```
array = dataset.values
X = array[:,0:-1]
Y = array[:,-1]
X_scaled = preprocessing.scale(X)
```

Figure 17 - preparing the data for the classifier

Now it is time to test each classifier and choose the best one. I made an array that contains each classifier and with a loop, tested each one. I used Kfold class of the Sklearn to divide the test and train data. The number of splits for Kfold is 10, and the random state is 7. You can see the result of the loop below. Random Forest Classifier has the best validation score, so I chose that one as my primary classifier to predict the model.

```
models=[]
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('RFC', RandomForestClassifier()))
```

```
results=[]
names=[]

for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=7, shuffle=True)
    cv_results = model_selection.cross_val_score(model, X_scaled, Y, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.845974 (0.003877)
LDA: 0.836632 (0.006512)
KNN: 0.857553 (0.006316)
CART: 0.935158 (0.004785)
NB: 0.471605 (0.018990)
SVM: 0.854579 (0.005671)
RFC: 0.949553 (0.002312)
```

Figure 18 - try each classifier

As mentioned above, the chosen classifier is Random Forest. “CL” is the variable that I assigned to the classifier class. I trained the classifier with the scaled data and “Y” as the target. Next part, I used this trained classifier to predict the model.

```
CL = RandomForestClassifier()
CL.fit(X_scaled, Y)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Figure 19 - most accurate classifier

Classification Techniques

This is the final part and we need to predict the model. First, I imported the pre-processed dataset as “test”.

```
test = pd.read_csv('readydatatest1.csv')
```

Figure 20 - read test dataset

The classifier needs data to be as arrays. So, first, I made the array and passed it to preprocessing class to scale it for the classifier. Then I gave the scaled data to the classifier and used the “predict_proba” function. This function predicts the probability of the target value and asked for the probability of 0. 0 represents lesser that $50K salary. In the end, I made a column in “test” dataset and passed the predicted probability to it.

```
arraytest = test.values
X_test = arraytest[:,1:]
X_scaled_test = preprocessing.scale(X_test)
```

```
prob = CL.predict_proba(X_scaled_test)[:,0]
```

```
test['Predicted'] = prob
```

Figure 21 - predict probability

At the end of the classification, I kept “ID” and “Predicted” columns and dropped all of the other columns. For the Kaggle submission, we only need these 2 columns. By extracting the predicted data, I finished this classification process.

```
test['Predicted'] = prob
```

```
test.drop(test.iloc[:, 1:33], inplace = True, axis = 1)
test
```

	ID	Predicted
0	21990	0.01
1	6594	0.01
2	18525	0.07
3	23747	0.04
4	37752	0.01
...	...	...
9995	34891	0.81
9996	15512	0.91
9997	27426	0.00
9998	23360	0.35
9999	3426	0.82

10000 rows × 2 columns

```
test.to_csv('Submission6.csv', index=False)
```

Figure 22 - extracting the predicted model

Summary

In this report, I briefly explained what I did to build the classifier. For pre-processing, I transformed 16 attributes of 38000 individuals into 33 columns and did binarizing and normalization for that part. For the classification, I tested 7 different classifiers and chose the most accurate one to build the classifier and predict the model of the test data set. I submitted the final predicted file to the Kaggle, and my score was **0.98020,** and my rank in the leader board was 14 at the time I too the following screenshot. You can see the details in the following figure.

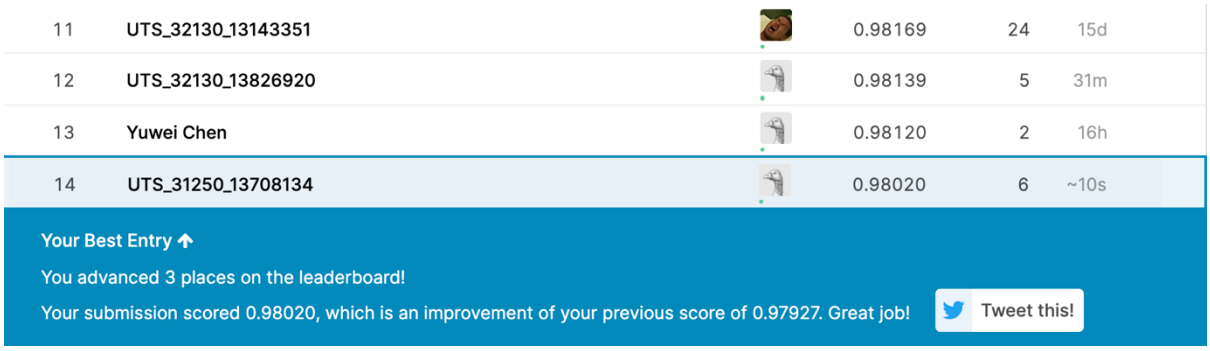

Figure 23 - Kaggle leader board